REMARKS

Rejection under 35 U.S.C §102

Claims 1 and 3 stand rejected under 35 U.S.C. 102(e) as being anticipated by U.S. Patent No. 6,523,027 (or '027) to Underwood.

Rejection of claim 1

In page 2 of the Action, the Examiner asserts that '027 discloses "a security protocol exchange in which at least some of the PDU passed between the parties are in a protocol format according to a self describing markup language". Applicant respectfully disagrees.

The '027 patent discloses a security protocol exchange, for example using the "secure sockets layer (SSL) certificate" of figure 16, and discloses the use of markup languages such as HTML, but fails to disclose using a markup language for security protocol exchange. Moreover, '027 recites (column 15, line 66 to column 16 line 3) *"HyperText Markup Language (HTML) to implement documents on the Internet together with a general-purpose secure communication protocol for a transport medium between the client and a company"*, and therefore makes plain that, according to '027, a secure protocol is distinct from an implementation using a Markup Language.

Furthermore, '027 actually teaches away from using a Markup Language such as HTML for a secure exchange purpose, for example by reciting (column 110, lines 65-67) that *"server side includes allow HTML authors to place commands inside their portion of the present descriptions that cause output to be modified whenever that portion of the present description is accessed by a user. Hackers can take advantage of server side includes if they are able to place arbitrary HTML statements on your server and then execute them"*. The skilled person reading the '027 patent would thus be cautioned away from using HTML because of its vulnerability to hacker attacks.

In view of the above, Applicant submits that '027 fails to disclose or suggest at least "security-protocol PDUs taking the form of electronic documents formatted

according to a self-describing markup language" as recited claim 1, and that claim 1 is patentable over '027.

Rejection of claim 3

Claim 3 has been amended and is now dependent on claim 1. At least for this reason, claim 3 is patentable over '027.

Rejection under 35 U.S.C §103

Claims 2, 4-6 stand rejected under 35 U.S.C. 103(a) as being unpatentable over '027. Claims 2, 4-6 are directly or indirectly dependent on claim 1. At least for this reason, claims 2, 4-6 are patentable over '027.

Patentability over references cited by the Applicant

The Applicant submits to the Examiner the enclosed excerpts from the opening chapter of the book "XML in Action" (William Pardi, Microsoft Press). Claim 1 recites a security protocol level that sits below the application layer, wherein the security-protocol PDUs (protocol data units) are expressed in the form of documents formatted according to a self-describing mark-up language. Application-level data is transferred simply as arbitrary data in a corresponding markup element of the PDU. The history of markup languages shows that they evolved to help the formatting of printed documents. The enclosed excerpts illustrate this and show that the use of markup languages has always been in respect of high-level documents at least partially intended for human use – that is, at an application-level in any layered view of a communication process. The claimed use of a markup language in a comprehensive security protocol residing below the application level is therefore a non-obvious departure from the prior uses of markup languages. Applicant submits that claim 1 is patentable over the enclosed reference.

* * *

Regarding the prior art made of record by the Examiner but not relied upon, Applicants believe that this art does not render the pending claims unpatentable.

In view of the above, Applicants submit that the application is now in condition for allowance and respectfully urge the Examiner to pass this case to issue.

The Commissioner is authorized to charge any additional fees that may be required or credit overpayment to deposit account no. 08-2025. In particular, if this response is not timely filed, the Commissioner is authorized to treat this response as including a petition to extend the time period pursuant to 37 CFR 1.136(a) requesting an extension of time of the number of months necessary to make this response timely filed and the petition fee due in connection therewith may be charged to deposit account no. 08-2025.

I hereby certify that this correspondence is being deposited with the United States Post Service with sufficient postage as first class mail in an envelope addressed to: Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on

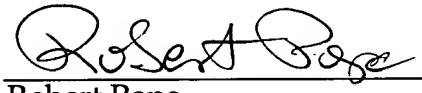
January 3, 2005
(Date of Transmission)

Corinda Humphrey
(Name of Person Transmitting)


(Signature)

January 3, 2005
(Date)

Respectfully submitted,



Robert Popa
Attorney for Applicants
Reg. No. 43,010
LADAS & PARRY
5670 Wilshire Boulevard, Suite 2100
Los Angeles, California 90036
(323) 934-2300 voice
(323) 934-0202 facsimile
rpopa@ladasparry.com

Attachments: Excerpts of the book "XML in action" (7 pages)

XML in Action

William J. Pardi

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Cataloging-in-Publication Data
Pardi, William J.
XML in Action / William J. Pardi.
p. cm.
ISBN 0-7356-0562-9
1. XML (Document markup language) I. Title.
QA76.76.H94P374 1999
005.7'2--dc21 98-52991
CIP

Printed and bound in the United States of America.

4 5 6 7 8 9 QMQM 4 3 2 1 0 9

Distributed in Canada by Penguin Books Canada Limited.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at mspress.microsoft.com.

Acquisitions Editor: Juliana Aldous
Project Editor: Thom Votteler
Technical Editor: Linda Harmony

It's a good guess that you're reading this book because you want to learn how to use XML (Extensible Markup Language). If you are like me, you want to pick up a computer book and start writing code by at least the second or third page. You've probably heard all the hype about how XML will change the Web and bridge gaps among the world's various types of digital information. You're convinced that XML is something you need to learn, and you might be anxious to jump right in and start coding. If you are that type of person, you won't have to wait too long. We'll get into some XML code before the end of the second chapter. But to really understand XML—and after all, that is the goal—you could probably benefit from some background information. These first few chapters provide a framework for the rest of the book—in addition to getting us into a little code. After reading these chapters, you should have a better understanding of and appreciation for XML. Establishing a framework is especially important with XML for a couple of reasons:

- You might not be familiar with some concepts utilized by markup languages. The information in these chapters will help get you up to speed on the basics of these languages and how they work.
- You might have experience using HTML (Hypertext Markup Language) or SGML (Standard Generalized Markup Language). You should understand how XML differs from these two languages and what makes it such a powerful alternative (or complement, depending on how you use it).

In many ways, XML represents a fundamental shift in the way information is delivered on the Web. While XML might not be as "flashy" as some of the other new Web technologies, it has the potential to have as much impact on Web delivery as HTML did several years ago. In this chapter, you'll begin to see why an extensible language like XML is necessary. We'll look at a brief background of text markup and how it works. We'll also examine differences between some of the more common markup languages.

A Brief History of Markup Languages

Markup has its roots in the print-based world, and even the term *markup* is a concatenation of the words *mark up*, which refer to the traditional way of marking up a document in the print and design worlds. The term *markup* specifically refers to tagging electronic documents for one of two purposes: to modify the look and formatting of text or to establish the structure and meaning of a document for output to some medium, such as a printer or the World Wide Web.

If you have worked with an HTML editor such as Microsoft FrontPage or a word processing program such as Microsoft Word, you are familiar with the idea of changing the formatting of text in a document. What you might not know is that those editing programs use markup to accomplish that formatting. We will look at how this works later in this chapter.

In addition to formatting text, markup can work to determine the structure and meaning (or context) of textual elements. For example, markup can establish that a document can contain only the elements Name, Birthday, and Age. It can further state that the document cannot contain Birthday and Age elements unless it contains a Name element. Markup can then state that the Name element must be text, the Birthday element must be a date, and the Age element must be a number. In this way, markup sets up the structure of the document and defines the semantic meaning of the elements. Later chapters will cover this subject in much more detail.

The Way Things Were

Before the advent of electronic publishing, a typed (or handwritten) document would be edited and marked up by hand on a draft copy. The draft would then go through several more rounds of revisions and reviews. Sometimes the document would be retyped, and sometimes it would end up with several layers of handwritten editorial marks. Based on a list of specifications for that type of document, formatting and style preferences for various parts of the document would be included as part of the handwritten notes. The document would then go to a typesetter, where the final typeset proof would be formatted and laid out. Then the finished document would be sent to the printer.

Enter Electronic Publishing

Electronic document preparation made a lot of this manual work unnecessary. It also made it much easier to change elements in a document at various points throughout the process before the document ever went to the printer. With traditional typesetting, such text formatting options as fonts, leading, margins, and justification were all established by the typesetter. The typesetter would use the typeface that was identified in the document mark up or in the specification sheet, perform copy-fitting calculations to make sure the page was readable, and then set the page in type. To accomplish this same level of control in the electronic world, a way to code the text was needed so that the output device would know how the document was supposed to be structured and how the text was supposed to look. The answer was electronic markup.

How Markup Works

Markup consists basically of codes, or *tags* (also called *tokens*), that are added to text to change the look or meaning of the tagged text. The tagged text for a document is usually called the *source code*, or just *code*, for that document. Most word processors, desktop publishing systems, and even simple text editors that can produce formatted text use some sort of markup language. For example, this book was written using Microsoft Word, which supports the markup language RTF (Rich Text Format).

Markup is commonly used to change the look of text by adding formatting, such as bold or italic fonts, text indents, font sizes, and font weights. Markup tags typically work by turning these attributes on when they are needed and off when they are not. Let's look at an example.

included the letters *DTD*. This line is called a *document type declaration*, and it tells the processor which DTD to use. DTDs will be discussed in Chapter 4, but if you are not familiar with them, all you need to know at this point is that each DTD works as a blueprint that defines a document structure.

Although each HTML document is *supposed* to conform to a DTD, in real-world applications, HTML processors (most often Web browsers) do not check a document against the DTD, nor do they even read the DTD. Because of this, most browsers let HTML authors break the rules a bit. For example, in the Memo.htm file shown previously, you could break the rules by putting the Title element outside the Head element or the Body element outside the HTML element. (You really shouldn't write code in this way, but the point is that you *could* do it and most browsers would still be able to read it.) As you come to understand XML, however, you will see that this casual approach to markup rules does not work when writing XML code.

NOTE

An *element* in many markup languages is simply a pair of opening and closing tags. For example, <TITLE> and </TITLE> are tags, but if you put them together, as in <TITLE></TITLE>, you have created a Title element. Most elements also contain some content between the opening and closing tags, as does the example on the previous page: *<TITLE>Memo</TITLE>*. However, not all markup languages require an opening tag and a closing tag to make up a valid element. In some cases, a single tag (usually the opening tag) is all that is needed.

Specific and Generalized Markup Languages

Two types of markup languages are in use today: *specific* markup and *generalized* markup. Specific markup languages are used to generate code that is specific to a particular application or device. These markup languages are often built to serve a particular need. Generalized markup describes the structure and meaning of the text in a document, but it does not define how the text should be used. In other words, the language itself is not made for any specific application and is generic enough to be used in many different applications. Documents described with a generalized markup language are usually portable to more applications than those described with a specific markup language. Let's examine these concepts in more detail.

Specific Markup Languages

The examples of markup shown so far in this chapter have demonstrated specific markup—that is, the HTML and RTF languages were developed for specific purposes. HTML has the specific purpose of formatting documents for the Web. And RTF has a similar purpose—that of text formatting. As you saw in the previous examples, however, the markup code for HTML and RTF look different, even though both were created for similar purposes. As you might have guessed, the two languages are not interchangeable. A processor that understands RTF will not understand HTML, and vice versa.

NOTE

Even though markup languages are not interchangeable, a single application might still be able to read and display several different kinds of markup. For example, with the correct filters installed, Word can read and display RTF, HTML, Plain Text, WordPerfect, Microsoft Works, and other types of documents. Note that different processing software is required for each markup language. Also, the markup codes are not interchangeable within a document. An RTF document, for example, must contain only RTF markup or the text will not display as intended.

Many markup languages have served quite well as document formatting tools for printing or for the Web. However, they do not perform as well at describing the data they contain or at providing contextual information for the data. Let's look at the markup for our RTF memo example again.

```
{\rtf1\ansi\ansicpg1252\deff0\deftab720{\fonttbl
  {\f0\fswiss MS Sans Serif;}
  {\f1\froman\fcharset2 Symbol;}
  {\f2\froman Times New Roman;}}
{\colortbl\red0\green0\blue0;}
\deflang1033\pard\plain\f2\fs20\b To: \plain\f2\fs20 Jodie
\par \plain\f2\fs20\b From: \plain\f2\fs20 Bill
\par \plain\f2\fs20\b Cc: \plain\f2\fs20 Philip
\par \plain\f2\fs20\b Subject: \plain\f2\fs20 Chapter 1
\par
\par What do you think of the format so far?
\par }
```

Notice that every tag describes how the text should be formatted but tells us nothing about the *kind* of text data included in the document. We could easily change all the text in the document and completely lose the fact that this was originally a memo document. We can do this because many markup languages are created for the specific purpose of describing text formatting and layout but not for any other purposes, such as defining a certain structure of data or providing a way to interchange incompatible data formats. Such specificity results in several limitations common to specific markup languages:

- Authors are limited to a particular set of tags. If this set of tags does not meet a need, authors must find a workaround or live with the limitation.
- A document might not be portable to other applications. Because the data is not self-describing, it cannot be used for any other purpose than that for which it was intended.
- The language probably has a proprietary way of marking up text that is not compatible with other markup languages. This can create confusion and extra work for authors who must use several languages to accommodate different applications.

NOTE

Self-describing documents, examined in later chapters, basically provide data about data (also called *metadata*) so that the data in the documents can stand apart from the formatting that describes how the documents are displayed. For example, a document might contain information in the form of a number. A self-describing document might identify the number as an age, the age as that of a tree, the tree as part of a reforestation project, and so on.

Back when electronic documents were starting to make a big impact on information delivery, it was obvious that these kinds of limitations would cause a lot of problems down the road. This encouraged the use of generalized markup languages.

Generalized Markup Languages

In the 1970s, Dr. C. F. Goldfarb (an attorney who eventually went to work for IBM) and two of his colleagues proposed a method of describing text that was not specific to an application or a device. The method had two basic parts:

- The markup should describe the structure of a document and not its formatting or style characteristics.
- The syntax of the markup should be strictly enforced so that the code can clearly be read by a software program or by a human.

The result of these suggestions was the Document Composition Facility Generalized Markup Language (DCF GML, or GML for short) developed for IBM. GML was the precursor to the Standard Generalized Markup Language (SGML) that was adopted as a standard by the International Organization for Standardization (ISO) in 1986.

NOTE

The ISO was founded in 1947 and comprises some 130 member countries. The ISO exists to "promote the development of standardization and related activities in the world with a view to facilitating the international exchange of goods and services, and to developing cooperation in the spheres of intellectual, scientific, technological and economic activity." The ISO's work results in a set of published standards that are used throughout the world. ISO standards affect fields ranging from telecommunications to agriculture to entertainment. You will often see a published standard referenced by its ISO number. (ISO 8879, for example, is the SGML standard.) For more information on the ISO, see *http://www.iso.ch/welcome.html*.

The SGML standard brought some important changes to text markup. In addition to providing a way to lay out the structure of a document, SGML added provisions for:

- Identifying the characters to be used in a document. This makes it easier to ensure that a processor can understand everything in a document by allowing a document to specify which character set it is using (ISO 646 or ISO 8859, for example).